A Dynamic to Static DSL Compiler for Image Processing Applications

Compilers for Parallel Computing 2016

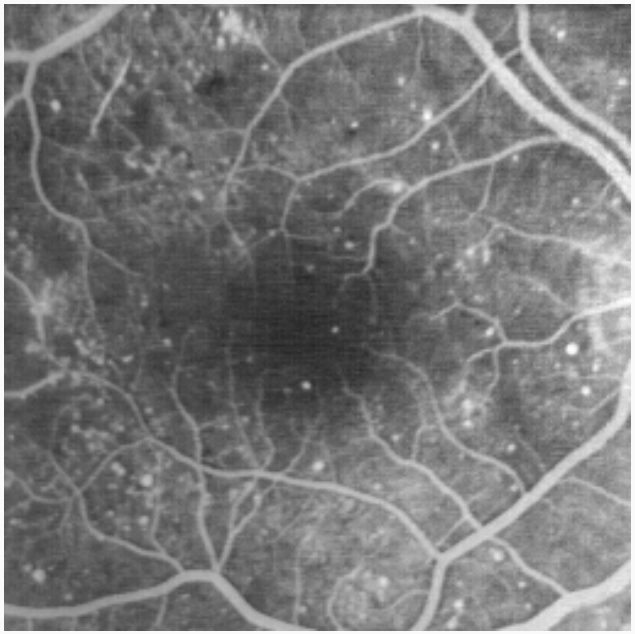
Pierre Guillou, Benoît Pin, Fabien Coelho, François Irigoin

July 8, 2016, Valladolid, Spain

MINES ParisTech, PSL Research University, France

Image processing applications

License plate detection



Retina analysis

Experiments with 7 image processing applications :

- anr999
- antibio
- burner
- deblocking
- licensePlate
- retina
- toggle

Image processing applications

License plate detection

Retina analysis

Experiments with 7 image processing applications :

- anr999
- antibio
- burner
- deblocking
- licensePlate
- retina
- toggle

Image processing applications development

Application developers

1. design a prototype in a high-level language (Python/MATLAB)
2. port and optimize manually to a set of hardware targets

Library developers

1. design a nice API
2. optimize for various hardware targets

Issue	mixed high and low level concerns in both cases
Objective	conciliate programmability and portability
Use case	SMIL and FREIA

The SMIL library

Simple (but efficient) Morphological Image Library [Fae11]

- new (2011) C++ image processing library
- targets modern processors
  - multi-cores — OpenMP
  - vector extensions — Loop auto-vectorization
- bindings — Python, Java, Ruby, GNU Octave (Swig)

```
import smilPython as smil

imin = smil.Image("input.png")  # read from disk
imout = smil.Image(imin)        # allocate imout
smil.dilate(imin, imout)        # morphological dilatation
imout.save("output.png")        # write to disk
```

FREIA framework

FREIA: FRamework for Embedded Image Applications [Bil+08]

- C image processing framework
- two-level API : atomic and complex image operators
- multiple hardware targets CPUs, GPUs, Manycores, FPGAs

FREIA optimizing compiler [CI13; GCI14]

- complex operators unfolding
- temporary variable elimination
- common sub-expression elimination
- backward/forward copy propagation
- target-specific code generation (operator aggregation, ...)
- ...

Morphological dilatation in FREIA

```c
#include "freia.h"

int main(void) {
  /* initializations... */

  /* image allocations */
  freia_data2d *imin = freia_common_create_data(/*...*/);
  freia_data2d *imout = freia_common_create_data(/*...*/);
  freia_common_rx_image(imin, /*...*/);    /* read from disk */
  /* morphological dilatation */
  freia_cipo_dilate(imout, imin, 8, 1);
  freia_common_tx_image(imout, /*...*/);   /* write to disk */
  /* freeing memory */
  freia_common_destruct_data(imin);
  freia_common_destruct_data(imout);

  /* shutdown... */
}
```

Bridging the gap

SMIL

- \+ high-level Python API
- \+ optimized ops on multicores
- − no other targets

FREIA

- − lower-level C API
- \+ optimized compilation stack
- \+ many targets

How to combine FREIA portability and SMIL programmability?

- port SMIL manually on every target — *expensive*
- re-implement SMIL using FREIA — *lose compilation stack*
- support SMIL in the FREIA compiler — *very expensive*
- convert SMIL Python app code into FREIA C

Bridging the gap

SMIL

- \+ high-level Python API
- \+ optimized ops on multicores
- − no other targets

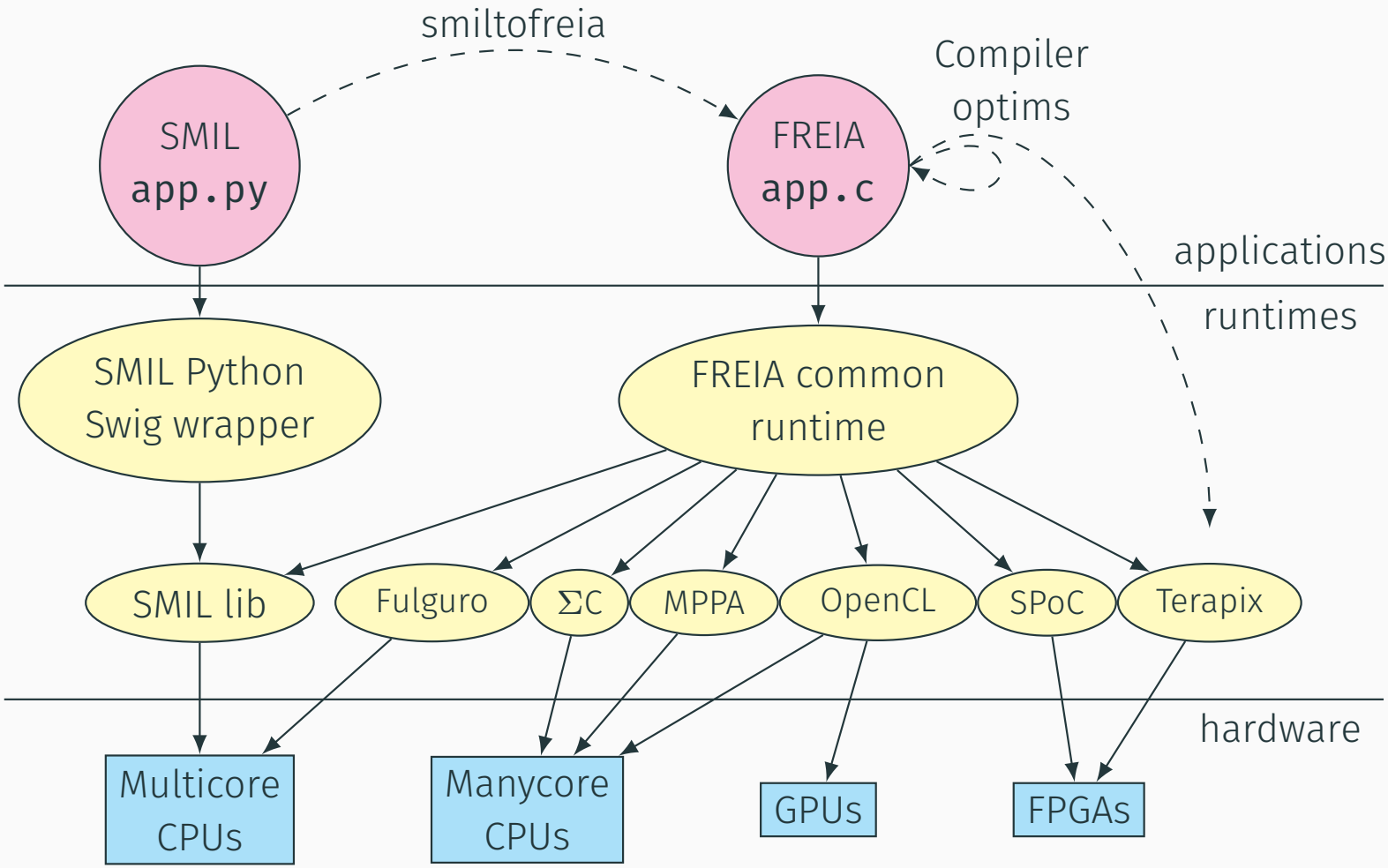
FREIA

- − lower-level C API
- \+ optimized compilation stack
- \+ many targets

How to combine FREIA portability and SMIL programmability?

- port SMIL manually on every target — *expensive*
- re-implement SMIL using FREIA — *lose compilation stack*
- support SMIL in the FREIA compiler — *very expensive*
- convert SMIL Python app code into FREIA C — `smiltofreia`

In summary


smiltofreia
SMIL
app.py
FREIA
app.c
Compiler
optims
applications
runtimes
SMIL Python
Swig wrapper
FREIA common
runtime
SMIL lib
Fulguro
ΣC
MPPA
OpenCL
SPoC
Terapix
hardware
Multicore
CPUs
Manycore
CPUs
GPUs
FPGAs

Generating directly FREIA C code

`smiltofreia`

- generate FREIA C code from the Python application AST
- written in Python
- transform every SMIL call in its FREIA equivalent
- takes care of memory management, variable declarations, etc.

Constraints on input code

- SMIL Python as a DSL
- variable types must be statically inferable

Dynamic to static compilation

Function polymorphism: canonical form

```
smil.dilate(i, o)
smil.dilate(i, o, 5)
```

```
smil.dilate(i, o, smil.SquSE(1))
smil.dilate(i, o, smil.SquSE(5))
```

Image expression atomization: temporary images

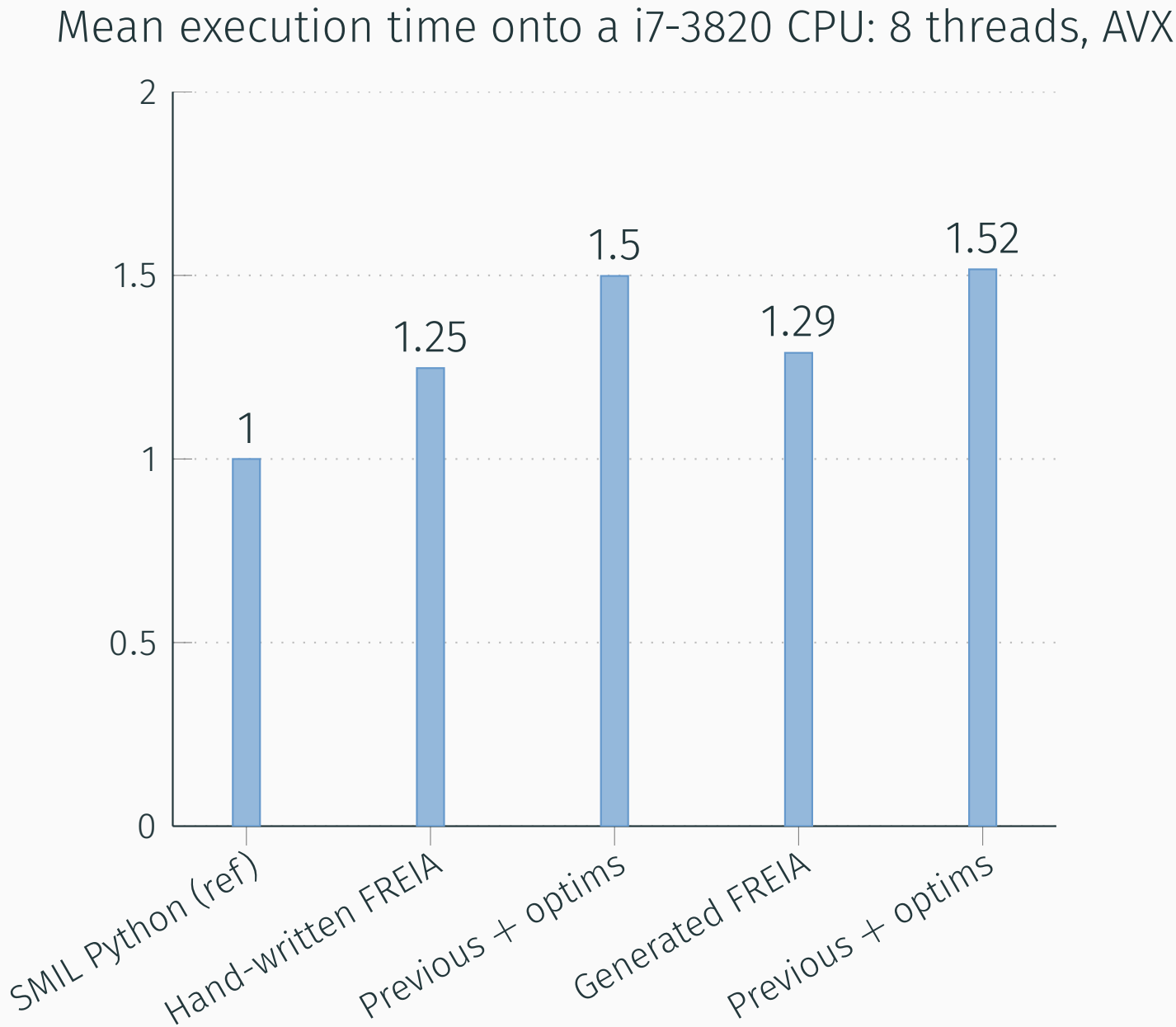
```
o = i0 * i1 + (i2 | i1)
```

```
freia_aipo_mul(tmp0, i0, i1);
freia_aipo_or(tmp1, i2, i1);
freia_aipo_add(o, tmp0, tmp1);
```

API variations: add new FREIA functions

```
freia_status freia_cipo_dilate_generic_8c(...);
freia_status freia_cipo_erode_generic_8c(...);
freia_status freia_cipo_gradient_generic_8c(...);
freia_status freia_aipo_mask(...);
```

Speedup on 7 image processing applications



Related work

Cython	ugly-compiles Python to C
Pythran	compiles scientific Python to C++/multicores+SIMD
Numba	is a Python-to-LLVM JIT compiler
Parakeet	targets CPUs and GPUs (CUDA)
Theano	optimizes Python linear algebra applications
Tensorflow	idem
Halide	is an image processing DSL compiler
PolyMage	idem

DSL compilation bring both programmability and portability

Key benefits of SMIL Python → FREIA C

- improved portability — *FREIA hardware targets*
- high programmability — *SMIL Python API*
- code reuse — *FREIA compilation stack*
- performance — *close to hand-written FREIA*

Future work

- increase API coverage
- experiment with larger SMIL applications

Thank you for your attention

Questions?

A Dynamic to Static DSL Compiler for Image Processing Applications

Compilers for Parallel Computing 2016

Pierre Guillou, Benoît Pin, Fabien Coelho, François Irigoin

July 8, 2016, Valladolid, Spain

MINES ParisTech, PSL Research University, France

References I

Matthieu Faessel. *SMIL: Simple (but efficient) Morphological Image Library*. 2011. URL: http://smil.cmm.mines-paristech.fr/.

Michel Bilodeau et al. *FREIA: FRamework for Embedded Image Applications*. French ANR-funded project with ARMINES (CMM, CRI), THALES (TRT) and Télécom Bretagne. 2008.

Fabien Coelho and François Irigoin. "API Compilation for Image Hardware Accelerators". In: *ACM Transactions on Architecture and Code Optimization* (Jan. 2013).

Pierre Guillou, Fabien Coelho, and François Irigoin. "Automatic Streamization of Image Processing Applications". In: *Languages and Compilers for Parallel Computing*. 2014.

Redbaron: Bottom-up approach to refactoring in python. URL: http://github.com/PyCQA/redbaron.

References II

Baron: a Full Syntax Tree library for Python. URL: https://github.com/PyCQA/baron.

Laurent Peuch. *RedBaron, une approche bottom-up au refactoring en Python*. Oct. 2014.

inspect — Inspect live objects. URL: https://docs.python.org/3/library/inspect.html.

ast — Abstract Syntax Trees. URL: https://docs.python.org/3/library/ast.html.

Alex Rubinsteyn et al. “Parakeet: A Just-In-Time Parallel Accelerator for Python”. In: Berkeley, CA: USENIX, 2012.

Serge Guelton et al. “Pythran: enabling static optimization of scientific Python programs”. In: *Computational Science & Discovery* (2015).

References III

Bryan Catanzaro, Michael Garland, and Kurt Keutzer. “Copperhead: Compiling an Embedded Data Parallel Language”. In: *16th ACM Symposium on Principles and Practice of Parallel Programming*. PPoPP ’11. 2011.

James Bergstra et al. “Theano: a CPU and GPU Math Expression Compiler”. In: *Python for Scientific Computing Conference (SciPy)*. Austin, TX, June 2010.

Christophe Clienti, Serge Beucher, and Michel Bilodeau. “A System On Chip Dedicated To Pipeline Neighborhood Processing For Mathematical Morphology”. In: *European Signal Processing Conference*. Aug. 2008.

Philippe Bonnot et al. “Definition and SIMD Implementation of a Multi-Processing Architecture Approach on FPGA”. In: *Design Automation and Test in Europe*. IEEE, Dec. 2008.

References IV

Benoit Dupont de Dinechin, Renaud Sirdey, and Thierry Goubier. “Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor”. In: *Procedia Computer Science 18*. 2013.

OpenCV: Open Source Computer Vision. URL: http://opencv.org/.

Matthieu Faessel and Michel Bilodeau. “SMIL: Simple Morphological Image Library”. In: *Séminaire Performance et Généricité, LRDE*. Villejuif, France, Mar. 2013. URL: https://hal-mines-paristech.archives-ouvertes.fr/hal-00836117.

Theodore Chabardes et al. “A parallel, O(n), algorithm for unbiased, thin watershed ”. working paper or preprint. Feb. 2016. URL: https://hal.archives-ouvertes.fr/hal-01266889.

References V

CMake: Build, Test and Package Your Software. URL: https://cmake.org/.

Swig: Simplified Wrapper and Interface Generator. URL: http://www.swig.org/.

OpenMP: Open Multi-Processing. URL: http://openmp.org/wp/.

The MPI Forum. *The Message Passing Interface*. URL: http://www.mpi-forum.org/.

François Irigoin, Pierre Jouvelot, and Rémi Triolet. “Semantical interprocedural parallelization: an overview of the PIPS project”. en. In: *Proceedings of ICS 1991*. ACM Press, 1991, pp. 244–251. ISBN: 0897914341. DOI: 10.1145/109025.109086. URL: http://portal.acm.org/citation.cfm?doid=109025.109086 (visited on 05/21/2014).

References VI

Christophe Clienti. *Fulguro image processing library*. Source Forge. 2008.

Khronos Group. *OpenCL: The open standard for parallel programming of heterogeneous systems*. URL: https://www.khronos.org/opencl/.

Cython: C-Extensions for Python. URL: http://cython.org/.

Thierry Goubier et al. “ΣC: A Programming Model and Language for Embedded Manycores”. In: 2011.

Pascal Aubry et al. “Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor.” In: *ICCS*. Ed. by Vassil N. Alexandrov et al. Vol. 18. Procedia Computer Science. Elsevier, 2013, pp. 1624–1633.

References VII

Auto-vectorization in GCC. URL: https://gcc.gnu.org/projects/tree-ssa/vectorization.html.

Herb Sutter. *Welcome to the Jungle*. 2011. URL: http://herbsutter.com/welcome-to-the-jungle/.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: ACM, 2015, 7:1–7:6. ISBN: 978-1-4503-4005-2. DOI: 10.1145/2833157.2833162. URL: http://doi.acm.org/10.1145/2833157.2833162.

M. Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *ArXiv e-prints* (Mar. 2016). arXiv: 1603.04467 [cs.DC].

References VIII

Jonathan Ragan-Kelley et al. “Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines”. In: *PLDI 2013* (June 2013), p. 12.

Ravi Teja Mullapudi, Vinay Vasista, and Uday Bondhugula. “PolyMage: Automatic Optimization for Image Processing Pipelines”. en. In: ACM Press, 2015, pp. 429–443. ISBN: 9781450328357. DOI: 10.1145/2694344.2694364. URL: http://dl.acm.org/citation.cfm?doid=2694344.2694364.

Backup slides

Morphological dilatation: `smiltofreia` output

```
#include "freia.h"
#include "smil-freia.h"

int main(int argc, char *argv[]) {
  /* initializations... */
  freia_data2d *imin;
  imin = freia_common_create_data(/* */);
  freia_data2d *imout;
  imout = freia_common_create_data(/* */);
#define e0 SMILTOFREIA_SQUSE
#define e0_s 1
  freia_cipo_dilate_generic_8c(imout, imin, e0, e0_s);
  freia_common_tx_image(imout, &fdout);
  freia_common_destruct_data(imout);
  freia_common_destruct_data(imin);
  /* shutdown... */
}
```

Refactoring Python code with RedBaron

Full Syntax Tree [Bar]

- AST + comments + formatting information
- `fst_to_code(code_to_fst(source_code)) == source_code`

RedBaron [Red; Peu14]

- an interface for manipulating a Python FST
- a refactoring tool

```
from redbaron import RedBaron

red = RedBaron("smil.dilate(imin, imout)")
for node in red.find_all("NameNode", value="imin"):
    node.value = "in"
print(red.dumps())  # smil.dilate(in, imout)
```

FST example: `smil.dilate(imin, imout)`


```
{"type": "atomtrailers",
 "value": [
     {"type": "name", "value": "smil"},
     {"type": "dot", "first_formatting": [],
      "second_formatting": []},
     {"type": "name", "value": "dilate"},
     {"first_formatting": [], "third_formatting": [],
      "type": "call", "fourth_formatting": [],
      "second_formatting": [],
      "value": [
          {"type": "call_argument",
           "first_formatting": [],
           "second_formatting": [], "target": {},
           "value": {"type": "name", "value": "imin"}},
          {"type": "comma", "first_formatting": [],
           "second_formatting": [{"type": "space",
                                  "value": " "}]},
          {"type": "call_argument", "first_formatting": [],
           "second_formatting": [], "target": {},
           "value": {"type": "name", "value": "imout"}}
     ]}]}
```

Compiling Python to C

Cython [Cyt]

- a tool for writing Python interfaces for C libraries
- a Python to C compiler

What we tried to do

1. develop a Cython wrapper around FREIA
2. convert SMIL Python code into FREIA/Cython with RedBaron
3. compile FREIA/Cython to C
4. apply FREIA compiler FAIL

Morphological dilatation: Cython C output

```
static PyObject *__pyx_pf_9smil_dilate_6Data2D_14cipoDilate(
    struct __pyx_obj_9smil_test_Data2D *__pyx_v_self,
    struct __pyx_obj_9smil_test_Data2D *__pyx_v_imout,
    __pyx_t_7pyfreia_int32_t __pyx_v_connexity,
    __pyx_t_7pyfreia_uint32_t __pyx_v_size) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations PyObject *__pyx_t_1 = NULL;
  __Pyx_RefNannySetupContext("cipoDilate", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = PyInt_FromLong(
      freia_cipo_dilate(__pyx_v_imout->_c_data2d,
                        __pyx_v_self->_c_data2d,
                        __pyx_v_connexity, __pyx_v_size));
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
```